Akita

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Akita engaged OtterSec to perform an assessment of the `akita` program.

This assessment was conducted between June 8th and June 17th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation.

We delivered final confirmation of the patches July 2nd, 2022.

Key Findings

The following is a summary of the major findings in this audit.

- 12 findings total
- 2 vulnerabilities which could lead to loss of funds:
  - OS-AKT-ADV-00: Missing loan recipient account check
  - OS-AKT-ADV-01: Improperly validated lender account

As a part of this audit, we also provided proofs of concept for each vulnerability to prove exploitability and enable simple regression testing. These scripts can be found at https://osec.io/pocs/akita. For a full list, see Appendix B.

02 | Scope

The source code was delivered to us in a git repository at https://github.com/otter-sec/akita/tree/master/programs/akita. This audit was performed against commit `425d73`.

There was a total of one program included in this audit. A brief description of the programs is as follows, and a full list of program files and hashes can be found in Appendix A.

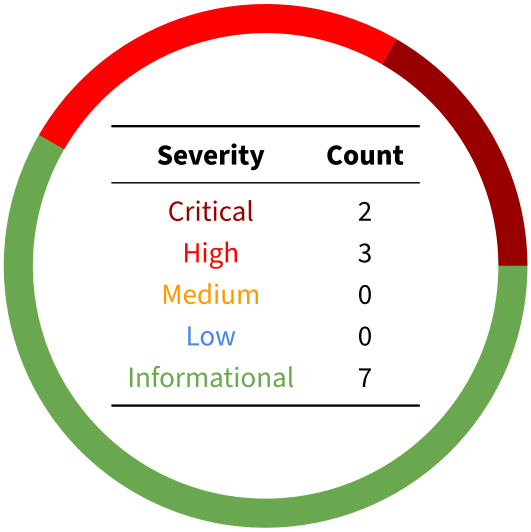
Name	Description
Akita	A peer-to-peer borrowing and lending protocol.

03 | Findings

Overall, we report 12 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact, and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.

The below chart displays the findings by severity.



Severity	Count
Critical	2
High	3
Medium	0
Low	0
Informational	7

Proofs of Concept

For each vulnerability we created a proof of concept to enable easy regression testing. We recommend integrating these as part of a comprehensive test suite. The proof of concept directory structure can be found in Appendix B.

A GitHub repository containing these proof of concepts can be found at https://osec.io/pocs/akita.

To run a PoC:

```sh
./run.sh <directory name>
```

For example,

```sh
./run.sh os-akt-adv-00
```

Each proof of concept comes with its own patch file, which modifies the existing test framework to demonstrate the relevant vulnerability. We also recommend integrating these patches into the test suite to prevent regressions.

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix E.

ID	Severity	Status	Description
OS-AKT-ADV-00	Critical	Resolved	Lender-controllable `loan_recipient_token_account` and `loan_token_mint` addresses lead to loss of funds.
OS-AKT-ADV-01	Critical	Resolved	A lender is able to close their external token account, preventing the borrower from repaying and forcing them to default.
OS-AKT-ADV-02	High	Resolved	Any user can lock the collateral on a borrow request forever by giving an unexpected `authority_bump` to the `AddCollateral` instruction.
OS-AKT-ADV-03	High	Resolved	Any user can add collateral with `collateral_amount` equal to 0 to a borrow request to prevent it from being seized.
OS-AKT-ADV-04	High	Resolved	Any user can add a collateral using a `collateral_mint` that they own and freeze the `collateral_token_account`.

OS-AKT-ADV-00 [crit] [resolved] | Missing Loan Recipient Account Check

Description

In the `AcceptBorrowRequest` instruction, the `loan_recipient_token_account` and `loan_token_mint` accounts are not properly checked against the `borrow_request` state.

This allows an attacker to transfer the requested tokens to their own account. Note that even though the borrower does not receive the tokens, they still are forced to repay them. Otherwise, they will lose their collateral to the lender.

Proof of Concept

Consider the following scenario:

1. A borrower creates a borrow request using the `InitializeBorrowRequest` instruction.
2. The borrower adds one or more collateral tokens to the borrow request using the `AddCollateral` instruction.
3. A malicious lender accepts the borrow request with their own `loan_token_mint` and `loan_recipient_token_account` addresses, using the `AcceptBorrowRequest` instruction.
4. Now, the borrow request is considered to be accepted by the lender, and the loan amount is given to the borrower. However, the borrower doesn't receive the loan in their account.
5. In addition, the borrower must now pay the requested borrow amount to the lender–or else they will lose their collateral to the lender.
6. After the duration of the loan has expired, if the borrow amount is not yet returned, the lender can seize the collateral using the `SeizeCollateral` instruction.

Remediation

Use Anchor constraints to enforce the missing checks in the `AcceptBorrowRequest` instruction.

RUST

```rust
borrow_request.loan_recipient_token_account == loan_recipient_token_account
    && borrow_request.loan_token_mint == loan_token_mint
```

Patch

Now using proper token account checks. Fixed in #2

OS-AKT-ADV-01 [crit] [resolved] | Improperly Validated Lender Account

Description

In the `AcceptBorrowRequest` instruction, the lender passes in an external token account which the borrower pays into. If the lender closes said account after accepting the borrow request, the borrower will become unable to repay the loan, and thus is forced to default.

Proof of Concept

Consider the following scenario:

1. A borrower creates a borrow request using the `InitializeBorrowRequest` instruction.
2. The borrower adds one or more collateral tokens to the borrow request using the `AddCollateral` instruction.
3. A malicious lender accepts the borrow request using the `AcceptBorrowRequest` instruction.
4. After accepting the borrow request, the lender closes their `repay_recipient_token_account`.
5. When the borrower tries to repay their loan using the `RepayLoan` instruction, the transaction fails, since the destination address of the transfer is closed
6. After the duration of the borrow request is over, the lender seizes the collateral using the `SeizeCollateral` instruction.

Remediation

To remediate this vulnerability, use a PDA that collects the repaid loan amount and implement another instruction for the lender to collect their loan amount from the PDA.

Patch

Now using a PDA to receive repaid loan amount. Fixed in #2.

OS-AKT-ADV-02 [high] [resolved] | Improper Collateral Bump Checks

Description

Using the AddCollateral instruction, any user can add collateral to a borrow request. In the instruction, authority_bump is taken as an input parameter and used to generate the borrow request authority PDA address.

If authority_bump is not equal to borrow_request.authority_bump, then the collateral cannot be withdrawn or seized. This is because the WithdrawCollateral and SeizeCollateral instructions use the borrow_request.authority_bump as bump to generate the borrow request authority PDA.

This leads to failure when withdrawing or seizing any collateral that was added before this collateral.

Proof of Concept

Consider the following scenario:

1. A borrower creates a borrow request using the InitializeBorrowRequest instruction.
2. The borrower adds one or more collateral tokens to the borrow request using the AddCollateral instruction.
3. A malicious user then adds more collateral using the AddCollateral instruction with authority_bump set to a value that is not equal to borrow_request.authority_bump.
4. Now, since the borrow request is still pending, if the borrower tries to withdraw the collateral, the transaction fails. Since this collateral cannot be withdrawn, all the collateral that is added before this collateral will be locked.
5. Even if a lender accepts the borrow request and tries to seize the collateral after the duration is complete, it will fail. This results from the SeizeCollateral instruction using the same authority_seeds macro as the WithdrawCollateral instruction.

Remediation

To remediate this vulnerability, add a check that makes sure borrow_request.authority_bump is equal to authority_bump to the AddCollateral instruction.

Patch

Added check: authority_bump == borrow_request.authority_bump. Fixed in #3

OS-AKT-ADV-03 [high] [resolved] | Improper Collateral Amount Checks

Description

Using the `AddCollateral` instruction, any user can add collateral to any borrow request. In the instruction, a check is not implemented to determine whether the `collateral_amount` is greater than 0.

If an amount of collateral equal to 0 tokens is added, then, after the duration of the borrow request is complete, the lender will not be able to seize any of the collateral that is part of the borrow request. This is a result of the constraint in `SeizeCollateral` instruction that checks whether the collateral amount is greater than 0. If it isn't, the instruction will fail. Since the current collateral cannot be seized, the previous collateral will also be locked from seizure.

Proof of Concept

Consider the following scenario:

1. A borrower creates a borrow request using the `InitializeBorrowRequest` instruction.
2. The borrower adds one or more collateral tokens to the borrow request using the `AddCollateral` instruction.
3. A malicious user then adds collateral using the `AddCollateral` instruction with `collateral_amount` equal to 0.
4. A lender accepts the borrow request and tries to seize the collateral after the duration of the borrow request is complete. This will fail as the `SeizeCollateral` instruction has a constraint on the `collateral_token_account` that determines if the amount in it is greater than 0.

Remediation

To remediate this vulnerabilty, add a check such as `collateral_amount > 0` to the `AddCollateral` instruction.

Patch

Only the borrower can add collateral. Fixed in #3

OS-AKT-ADV-04 [high] [resolved] | Unsound collateral design

Description

Using the `AddCollateral` instruction, any user can add collateral to any borrow request. A malicious user can add collateral with a `collateral_mint` of their own and a `freeze_authority` set to their own account.

The user can then freeze the `collateral_token_account` using their freeze authority. Then neither the borrower nor the lender will be able to withdraw or seize the collateral that is added prior to the malicious collateral, since the token transfer instruction in `WithdrawCollateral` and the `SeizeCollateral` instruction will fail.

Proof of Concept

Consider the following scenario:

1. A borrower creates a borrow request using the `InitializeBorrowRequest` instruction.
2. The borrower adds one or more collateral tokens to the borrow request using the `AddCollateral` instruction.
3. A malicious user then adds collateral using the `AddCollateral` instruction with a `collateral_mint` of their own and a source token account. The tokens are transferred from source token account to the token supply (PDA).
4. The malicious user now freezes the token supply PDA account.
5. This prevents both the borrower and the lender from taking the collateral that is added prior to the malicious collateral.

Remediation

To remediate this vulnerability, implement the `WithdrawCollateral` and `SeizeCollateral` instructions such that they are independent of their order. For example, one could take the index of the collateral and use it to generate the `collateral_token_account` PDA in the `WithdrawCollateral` and `SeizeCollateral` instructions.

Patch

Changed so that only the borrower can add collateral. Fixed in #3

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they do represent antipatterns, and could introduce a vulnerability in the future.

ID	Status	Description
OS-AKT-SUG-00	Resolved	Close the edit proposal accounts to save the rent amount for the proposer.
OS-AKT-SUG-01	Resolved	Bumps are unnecessarily taken as input parameters.
OS-AKT-SUG-02	Resolved	An `AcceptBorrowRequest` instruction can be modified by the borrower after the user accepts.
OS-AKT-SUG-03	Resolved	The duration parameter in `borrow_request.params` should be checked to ensure it is greater than 0.
OS-AKT-SUG-04	Resolved	The `InitializeFeeReceiver` instruction can be replaced by using the associated token account.
OS-AKT-SUG-05	Resolved	The fee receiver receives less fee due to floor division in fee calculation.
OS-AKT-SUG-06	Resolved	Using a constraint to check the borrow request of an edit proposal in the `AcceptEditProposal` instruction.

OS-AKT-SUG-00 [resolved] | Closing Edit Proposals

Description

The `edit_proposal` PDA accounts are created by taking the rent amount from the proposer. Those accounts should be properly closed and the rent amount should be sent back to the proposer.

Remediation

The `edit_proposal` accounts should be closed as part of the `AcceptEditProposal` instruction and the rent amount sent back to the proposer.

Alternatively, add a `CancelEditProposal` instruction to close the edit proposals that are not accepted. The rent expended there should be sent back to the proposer as well.

Patch

Added a new instruction to close `EditProposal` in 0e9d501

OS-AKT-SUG-01 [resolved] | Unnecessary Use of Bumps

Description

The token_supply_bump in the WithdrawCollateral and SeizeCollateral instructions is unnecessary. The fee_receiver_bump and market_bump in the WithdrawFee and AcceptBorrowRequest instructions are unnecessary. The market_bump in the InitializeMarket instruction is unnecessary.

Remediation

The unnecessary bumps taken as input parameters should be removed.

Patch

Removed unnecessary bumps in 55edbda

OS-AKT-SUG-02 [resolved] | Missing AcceptBorrowRequest Slippage Checks

Description

The AcceptBorrowRequest instruction has no way to check for changes between accepting the request on the frontend, and submitting the transaction. For example, the borrower could reduce the interest, increase the duration, or change the borrow amount.

Remediation

Take the borrow request parameters and collateral data from the lender as input data in the AcceptBorrowRequest instruction and verify whether they are same as the borrow_request.params and associated collateral.

Patch

Accept a new parameter in AcceptBorrowRequest that double checks borrow request parameters in bf16d82

OS-AKT-SUG-03 [resolved] | Borrow request duration should not be 0

Description

The duration parameter in `borrow_request.params` should be checked to ensure it is greater than 0.

Remediation

The check `duration > 0` should be added to the `is_valid` function in the borrow request parameters.

Patch

Added a `duration > 0` check in `is_valid` function of borrow request parameters in f61fe72

OS-AKT-SUG-04 [resolved] | Use Associated Token Account

Description

Instead of using the `InitializeFeeReceiver` instruction to create a PDA for the fee receiver token account, an associated token account can be used in the `AcceptBorrowRequest` instruction with `associated_token::authority` set to `akita_authority`. This way there is no need to initialize a fee receiver PDA token account.

Remediation

Use an associated token account in the `AcceptBorrowRequest` instruction.

```rust
pub struct AcceptBorrowRequest<'info> {
    // ...

    #[account(
        seeds = [ b"Authority" ],
        bump
    )]
    /// CHECK: we don't do checking since it is authority.
    pub akita_authority: UncheckedAccount<'info>,

    #[account(
        init_if_needed,
        payer = payer,
        associated_token::mint = loan_token_mint,
        associated_token::authority = akita_authority,
    )]
    pub fee_receiver: Box<Account<'info, TokenAccount>>,

    #[account(mut)]
    pub payer: Signer<'info>,

    // ...
}
```

Patch

Akita acknowledges the finding but doesn't believe it has security implications. However, they may deploy a fix to address it.

OS-AKT-SUG-05 [resolved] | Calculating Borrow Fee Does Floor Division by Default

Description

The `calculate_borrow_fee` function in `state.rs` is used to calculate the fee that is sent to the fee receiver. The `checked_div` by default floors the decimal value, which results less tokens being sent as the fee when the division results in fractional numbers.

Consider the following values for calculating fee:

```
expected_repay_amount = 10001
borrow_amount = 10000
borrow_fee_bips = 1000
interest = 10009 - 10000 = 9
fee = (interest * borrow_fee_bips) / 10000
fee = (9 * 1000) / 10000 = 9000 / 10000 = 0.9
```

Since the fee is floored, the 0.9 token fee will become 0.

Remediation

Use ceiling division via `spl_math::checked_ceil_div` instead of `checked_div`.

Patch

Akita acknowledges the finding but doesn’t believe it has security implications. However, they may deploy a fix to address it.

OS-AKT-SUG-06 [resolved] | Checking Borrow Request for Edit Proposal

Description

A constraint to check for the `borrow_request` of an edit proposal in `AcceptEditProposal` instruction should be added to ensure that the edit proposal belongs to the given borrow request. Note that this is already implicitly enforced through the PDA seeds, but this additional constraint would be good to implement as defense in depth.

Remediation

Add the constraint `edit_proposal.borrow_request == borrow_request.key()` as a check in the `AcceptEditProposal` instruction.

Patch

Added an additional constraint in ac2637a.

A | Program Files

Below are the files in scope for this audit and their corresponding SHA256 hashes.

```
akita/
  Cargo.toml                                              1e2cded07bc35570f4b5d88e2ea92087c43e1302
  Xargo.toml                                              815f2dfb6197712a703a8e1f75b03c6991721e9e
  src/
    events.rs                                             3dd336df765f2158fc6b824e3bdfbe8b258896c8
    lib.rs                                                030df9ddaae281f83c05a2d11b4413e097d35c53
    macros.rs                                             5ba6f1f41ebe3e357b6b229fdf9b3539ceb435af
    state.rs                                              ed604e3458abffd9a371acd8d616a2de87a6379b
    instructions/
      accept_borrow_request.rs                            b3f66f71d28de196fbddd2d963b5099a79ea7c12
      accept_edit_proposal.rs                             273eca53ad5bd69bdbb60e6671363aad96018a12
      add_collateral.rs                                   5af7e966df2a1daee30b766018d963adced56d4e
      close_borrow_request.rs                             7ae3f4846abbe5a83afc4f2f885e375397690b40
      create_edit_proposal.rs                             2aed5d5b9d2be27c85b7b991bdf82ea95775e2d6
      initialize_borrow_request.rs                        c11447bc7292d0639eb573ea42a92710c23854ab
      initialize_fee_receiver.rs                          c9f5528655788e1622cb90eccb4b8ffc6752ea72
      initialize_market.rs                                5102a80afc4388276fdc7adc3d57ca1bc51cf30d
      mod.rs                                              0f396bd207bb7b602d3561833d155215c993fe00
      repay_loan.rs                                       6402bd51d3c840122721a70108bf515294ac2fdd
      seize_collateral.rs                                 54eab105bf520bd674000dcc9a346357c2335e30
      withdraw_collateral.rs                              b607a3dbfc204380bf2409de5b676622843ba0ce
      withdraw_fee.rs                                     8fc3365a614add72be57934599af5f1caf90324e
      withdraw_repaid_fund.rs                             48b27170450bec307c2390620a839aaabbad93eb
```

B | Proof of Concepts

Below are the provided proof of concept files and their corresponding SHA256 hashes.

```
pocs/
  os-akt-adv-00/
    hash                                       9bce042b4c8c705ab29258971ac7fade5199c02a
    patch                                      02613bc6c37081f38f54a653a41a458cd3cdf7ab
    run.sh                                     ade30af485cbf5d780ccb2d9997cc5e0288ae3fe
  os-akt-adv-01/
    hash                                       9bce042b4c8c705ab29258971ac7fade5199c02a
    patch                                      e3f0d5e5edbe8c9a537aa0731b99e5946e6cfa4f
    run.sh                                     ade30af485cbf5d780ccb2d9997cc5e0288ae3fe
  os-akt-adv-02/
    hash                                       9bce042b4c8c705ab29258971ac7fade5199c02a
    patch                                      797d27197f1f36274e7fa67c90688d0d37ff59a1
    run.sh                                     ade30af485cbf5d780ccb2d9997cc5e0288ae3fe
  os-akt-adv-03/
    hash                                       9bce042b4c8c705ab29258971ac7fade5199c02a
    patch                                      bfeea43f59ee1b96688638f08a752266fb0b2f15
    run.sh                                     ade30af485cbf5d780ccb2d9997cc5e0288ae3fe
  os-akt-adv-04/
    hash                                       9bce042b4c8c705ab29258971ac7fade5199c02a
    patch                                      f2e65980caee707fcdcb0681a861b52a64e77d1d
    run.sh                                     ade30af485cbf5d780ccb2d9997cc5e0288ae3fe
```

C | Implementation Security Checklist

Unsafe arithmetic

Integer underflows or overflows	Unconstrained input sizes could lead to integer over or underflows, causing potentially unexpected behavior. Ensure that for unchecked arithmetic, all integers are properly bounded.
Rounding	Rounding should always be done against the user to avoid potentially exploitable off-by-one vulnerabilities.
Conversions	Rust as conversions can cause truncation if the source value does not fit into the destination type. While this is not undefined behavior, such truncation could still lead to unexpected behavior by the program.

Account security

Account Ownership	Account ownership should be properly checked to avoid type confusion attacks. For Anchor, the safety of unchecked accounts should be clearly justified and immediately obvious.
Accounts	For non-Anchor programs, the type of the account should be explicitly validated to avoid type confusion attacks.
Signer Checks	Privileged operations should ensure that the operation is signed by the correct accounts.
PDA Seeds	PDA seeds are uniquely chosen to differentiate between different object classes, avoiding collision.

Input validation

Timestamps	Timestamp inputs should be properly validated against the current clock time. Timestamps which are meant to be in the future should be explicitly validated so.
Numbers	Sane limits should be put on numerical input data to mitigate the risk of unexpected over and underflows. Input data should be constrained to the smallest size type possible, and upcasted for unchecked arithmetic.
Strings	Strings should have sane size restrictions to prevent denial of service conditions
Internal State	If there is internal state, ensure that there is explicit validation on the input account's state before engaging in any state transitions. For example, only open accounts should be eligible for closing.

Miscellaneous

Libraries	Out of date libraries should not include any publicly disclosed vulnerabilities
Clippy	cargo clippy is an effective linter to detect potential anti-patterns.

D | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an onchain program. In other words, there is no way to steal tokens or deny service, ignoring any Solana specific quirks such as account ownership issues. An example of a design vulnerability would be an onchain oracle-which could be manipulated by flash loans or large deposits.

On the other hand, auditing the implementation of the program requires a deep understanding of Solana's execution model. Some common implementation vulnerabilities include account ownership issues, arithmetic overflows, and rounding bugs. For a non-exhaustive list of security issues we check for, see Appendix C.

Implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions, both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach any target in a team of two. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

E | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities which immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority/token account validation
- Rounding errors on token transfers

High

Vulnerabilities which could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities which could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input cause computation limit exhaustion
- Forced exceptions preventing normal use

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation
- Uncaught Rust errors (vector out of bounds indexing)